Does this program crash?

```
def foo():
  x = input()
  f(x)
```

```
def f(x):
  if (x > 0):
    f(x-1)
  else:
    g(x)

def g(x):
  1/x
```

Yes! When input = 0!

How do we **prove** this?

```
def foo():
  x = input()
  f(x)
```

```
def f(x):
  if (x > 0):
    f(x-1)
  else:
    g(x)

def g(x):
  1/x
```

```
def foo():
    x = 1
    f(x)
```

```
def f(x):
    if (x > 0):
        f(x-1)
    else:
        g(x)

def g(x):
    1/x
```

Could try **testing**

```
def foo():
  x = 2
  f(x)
```

Could try **testing**

```
def f(x):
  if (x > 0):
    f(x-1)
  else:
    g(x)

def g(x):
  1/x
```

```
def foo():
    x = 3
    f(x)
```

Could try **testing**

```
def f(x):
    if (x > 0):
        f(x-1)
    else:
        g(x)

def g(x):
    1/x
```

```
def foo():
    x = 4
    f(x)
```

Could try **testing**

```
def f(x):
    if (x > 0):
        f(x-1)
    else:
        g(x)

def g(x):
    1/x
```

```
def foo():
  x = 4
  f(x)
```

```
def f(x):
  if (x > 0):
    f(x-1)
  else:
    g(x)

def g(x):
  1/x
```

Testing doesn’t prove absence of bugs!

Instead: **simulate** program with “abstract” inputs

+, -, 0

```
def foo():
  x = +
  f(x)
```

```
def f(x):
  if (x > 0):
    f(x-1)
  else:
    g(x)

def g(x):
  1/x
```

+ - 1 could result in 0!

Instead: **simulate** program with “abstract” inputs

+, -, 0

```
def foo():
  x = +
  f(x)
```

```
def f(x):
  if (x > 0):
    f(x-1)
  else:
    g(x)

def g(x):
  1/x
```

Now consider f(0)…

This process is called **abstract interpretation**

Approximate **infinite** behavior w/ **finite** runs

Many applications:

- Optimization
- Proving properties
- Enforcing security

Detour: Information Flow

Which of the following leaks the input?

```
void foo(secret) {
    send(secret);
}
```

Detour: Information Flow

Which of the following leaks the input?

```
void foo(secret) {
    send(secret);
}
```

If Barb knows the program, and sees some value *v*, she knows the secret was *v*

Detour: Information Flow

Which of the following leaks the input?

```
void foo(secret) {
    send(secret);
}
```

```
void bar(secret) {
    send(0);
}
```

Detour: Information Flow

Which of the following leaks the input?

```
void foo(secret) {
    send(secret);
}
```

```
void bar(secret) {
    send(0);
}
```

Barb sees 0, can't learn secret

Detour: Information Flow

Which of the following leaks the input?

```
void baz(secret) {
    if(secret == 0) {
      send(0);
    } else {
      send(1);
    }
}
```

Detour: Information Flow

Which of the following leaks the input?

```
void baz(secret) {
    if(secret == 0) {
      send(0);
    } else {
      send(1);
    }
}
```

Barb sees 0, knows secret == 0, otherwise knows it was != 0

Detour: Information Flow

Which of the following leaks the input?

```
void baz(secret) {
    if(secret == 0) {
      send(0);
    } else {
      send(1);
    }
}
```

Barb sees 0, knows secret == 0, otherwise knows it was != 0

This is called an *implicit flow*

(Information leaked via control path)

Unfortunately, **no popular languages** readily enable enforcing information flow…

But possible to **verify** using abstract interpretation!

Current work…

Scale program analyses to **supercomputers**

Verify security properties via program analysis

Building **new** languages w/ security built in from start